Create account Log in

Article Talk Read Edit View history Search

WIKIPEDIA
The Free Encyclopedia

Navigation
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Mainline DHT

From Wikipedia, the free encyclopedia

This article **relies on references to primary sources**. Please add references to secondary or tertiary sources. *(December 2012)*

Mainline DHT is the name given to the Kademlia-based Distributed Hash Table (DHT) used by BitTorrent clients to find peers via the BitTorrent protocol. The idea of utilizing a DHT for distributed tracking was first implemented[1] in Azureus 2.3.0.0 (now known as Vuze) in May 2005, from which it gained significant popularity. BitTorrent, Inc. then incorporated a similar DHT into their client, called Mainline DHT and thus popularized the use of distributed tracking in the BitTorrent Protocol. Recent measurement shows by 2013 users of Mainline DHT is from 10 million to 25 million, with a daily churn of at least 10 million.[2]

Contents

1 Description
1.1 Operation
1.1.1 Token
1.2 KRPC
1.2.1 Queries
1.2.2 Responses
1.2.3 Errors
1.3 Routing Table
2 BitTorrent Protocol Extension
2.1 Torrents
3 Implementations
4 References
5 External links

Description [edit]

Mainline DHT is based on the popular Kademlia DHT design and is described in a draft published here . Previous to usage of a DHT for distributing peers, trackers were the only method of finding peers. The key feature of using the DHT over trackers is that the decentralized approach favours the nature of the BitTorrent protocol. The DHT operates by distributing lists of peers identified by the SHA-1 hash of the torrent.

Operation [edit]

The SHA-1 hash of a torrent, the **infohash**, is synonymous with a Kademlia key, which is used for finding peers (values) in the overlay network. To find peers in a swarm, a node sends a *get_peers* query with the infohash as key (equivalent to a Kademlia *FIND_VALUE*) to the closest known nodes (with respect to the key distance). Like Kademlia, if the node does not return the value (peers), it persists further in an iterative operation. However, after the search is exhausted, the client then also inserts the peer contact information for itself onto the responding nodes with IDs closest to the infohash of the torrent.

Token [edit]

Nodes use an additional measure known as a **token** to ensure others don't sign up other hosts for torrents. The return value for a query for peers includes this opaque value. For a node to announce that its controlling peer is downloading a torrent, it must present the token received from the same queried node in a recent query for peers. When a node attempts to "announce" a torrent, the queried node checks the token against the querying node's IP address.

Mainline DHT uses the SHA1 hash of the IP address concatenated onto a secret that changes every five minutes for a token value. Tokens up to ten minutes old are accepted.

KRPC [edit]

A node in the Mainline DHT consists of an IP and port combination. Nodes communicate via an RPC protocol - KRPC. KRPC is a simple protocol that consists of nodes sending messages (queries, replies and errors) containing BEncoded dictionaries over UDP.

A KRPC message is a single dictionary with two keys common to every message and additional keys depending on the type of message. Every message has a key *"t"* with a string value representing a transaction ID. This transaction ID is generated by the querying node and is echoed in the response, so responses may be correlated with multiple queries to the same node. The transaction ID should be encoded as a short string of binary numbers, typically 2 octects are enough as they cover 2^16 outstanding queries. The other key contained in every KRPC message is *"y"* with a single character value describing the type of message. The value of the "y" key is one of *"q"* for query, *"r"* for response, or *"e"* for error.

Queries [edit]

Queries, or KRPC message dictionaries with a "y" value of "q", contain two additional keys; "q" and "a". Key "q" has a string value containing the method name of the query. Key "a" has a dictionary value containing named arguments to the query.

Responses [edit]

Responses, or KRPC message dictionaries with a "y" value of "r", contain one additional key "r". The value of "r" is a dictionary containing named return values. Response messages are sent upon successful completion of a query.

Errors [edit]

Errors, or KRPC message dictionaries with a "y" value of "e", contain one additional key "e". The value of "e" is a list. The first element is an integer representing the error code. The second element is a string containing the error message. Errors are sent when a query cannot be fulfilled.

Routing Table [edit]

Buckets are structured differently from those in Kademlia. Instead of a list of 160 buckets, BitTorrent starts with only one bucket. When a bucket becomes full, one of two things can happen:

1. The bucket is split
2. Old nodes are pinged (like in Kademlia)

Splitting is an operation that occurs only if our own node ID falls within the range of the bucket. The bucket being split is replaced by two new buckets each with half the range of the old bucket and the nodes from the old bucket are distributed among the two new ones.

There are 2 benefits to this bucket implementation:

- Less memory is used for a routing table of less than 160 buckets
- When searching buckets, it isn't necessary to retrieve additional nodes from adjacent buckets because it is guaranteed that there are enough in the current bucket

BitTorrent Protocol Extension [edit]

The BitTorrent protocol has also been extended to exchange node UDP port numbers between peers that are introduced by a tracker. In this way, clients can get their routing tables seeded automatically through the download of regular torrents. Newly installed clients who attempt to download a trackerless torrent on the first try will not have any nodes in their routing table and will need the contacts included in the torrent file.

Peers supporting the DHT set the last bit of the 8-byte reserved flags exchanged in the BitTorrent protocol handshake. Peer receiving a handshake indicating the remote peer supports the DHT should send a PORT message. It begins with byte 0x09 and has a two byte payload containing the UDP port of the DHT node in network byte order. Peers that receive this message should attempt to ping the node on the received port and IP address of the remote peer. If a response to the ping is received, the node should attempt to insert the new contact information into their routing table according to the usual rules.

Torrents [edit]

A trackerless torrent dictionary does not have an *"announce"* key. Instead, a trackerless torrent has a *"nodes"* key, which functions as a list of Bootstrapping nodes (in case we haven't already joined the overlay network). This key is normally set to the *K* closest nodes in the torrent generating client's routing table.

A *"private"* flag has also been unofficially introduced, telling clients to restrict the use of decentralized tracking regardless of the user's desires. The flag is intentionally placed in the info section of the torrent so that it cannot be disabled or removed without changing the identity of the torrent. The purpose of the flag is to prevent torrents from being shared with clients that do not have access to the tracker.

Implementations [edit]

Mainline DHT was first included in version 4.2.0 of BitTorrent (software) (November 2005). Since then, it has been implemented by a number of other clients:

- µTorrent
- Transmission
- rTorrent
- KTorrent
- BitComet
- Deluge
- BitSpirit
- Vuze with the MlDHT plugin
- Shareaza
- Tixati

Libdht is a free library that provides a client-independent implementation of the mainline DHT. It is used notably by Transmission and Shareaza.

References [edit]

1. ^ "Vuze Changelog" . Azureus.sourceforge.net.
2. ^ Wang, Liang; Kangasharju, Jussi. (2013). "Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT". *IEEE Peer-to-Peer*. Retrieved 31 September 2013.

External links [edit]

- Official specification and extensions for IPv6 support .
- Mainline DHT Measurement

Wikimedia Commons has media related to ***BitTorrent***.

Categories: BitTorrent clients

This page was last modified on 27 March 2014 at 13:18.

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Privacy policy About Wikipedia Disclaimers Contact Wikipedia Developers Mobile view